The Aftermath of a Fuzz Run:
What to do with all those crashes?

David Moore

Embedded Linux Conference

David Moore Bio

NeXT, Apple, Weblogic,
BEA Systems,
Azul Systems

Google, Twitter, Netflix, Optimizely, Card, kernel, ruby, php, cpio

FuzzStatiOn

Founder/CEO

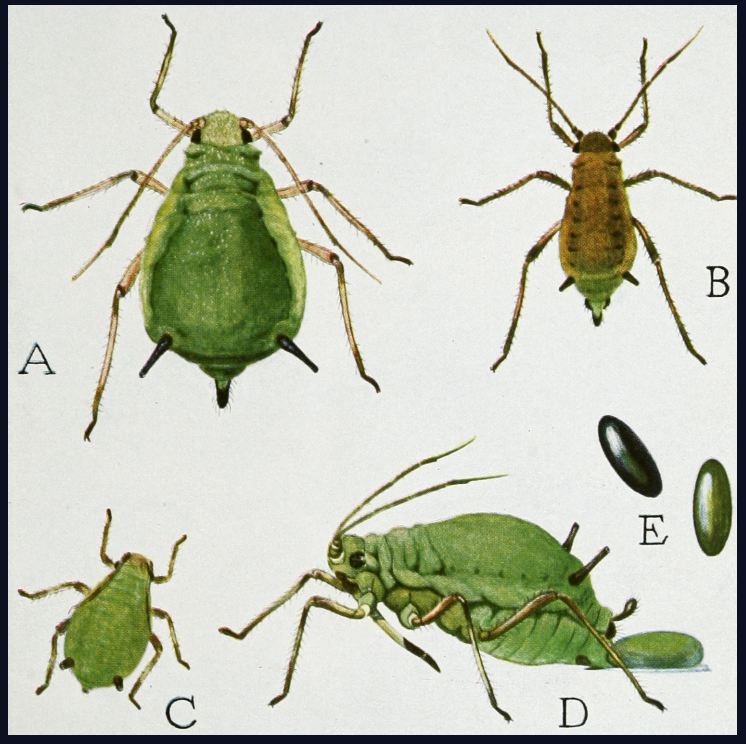
Talk Outline

1> Introduce/Review Memory Corruption Bugs

2> A Post Fuzz Run Workflow

3> Real World Examples

Section 1a:
Introduce / Review Memory Corruption Bugs



Invalid Reads/Writes

Stack vs Heap Corruption

```
int main (int argc, char **argv) {
    char buf[8];
    strcpy(buf, argv[1]);
}
```

```
./a.out AAAAAAAAAAAAAAA
```

Use After Free

```
char* x = (char*)malloc(4);
...
free(x);
...
printf(x)   // uaf
```

Other Memory Bugs

Section 1b:
What is Exploitability?

Re-programming with input data - not code

Re-programming with existing code in the process

Does “exploitability” matter?

Exploitable By Whom?

Google Project Zero

NSA

Many modern exploits are bug chains

Surprisingly Exploitable

C-Ares / Chrome OS Remote Code Execution

The main Google bug for this problem is still not open since they still have pending mitigations to perform, but since the c-ares issue has been fixed I've been told that it is fine to talk about this publicly.

c-ares writes a 1 outside its buffer

c-ares has a function called ares_create_query. It was added in 1.10 (released in May 2013) as an updated version of the older function ares_mkquery. This detail is mostly interesting because Google uses an older version than 1.10 of c-ares so in their case the flaw is in the old function. This is the two functions that contain the problem we're

C-ARES, SECURITY

A SINGLE BYTE WRITE OPENED A ROOT EXECUTION EXPLOIT

OCTOBER 14, 2016 DANIEL STENBERG 7 COMMENTS

Thursday, September 22nd 2016. An email popped up in my inbox.

> *Subject: ares_create_query OOB write*

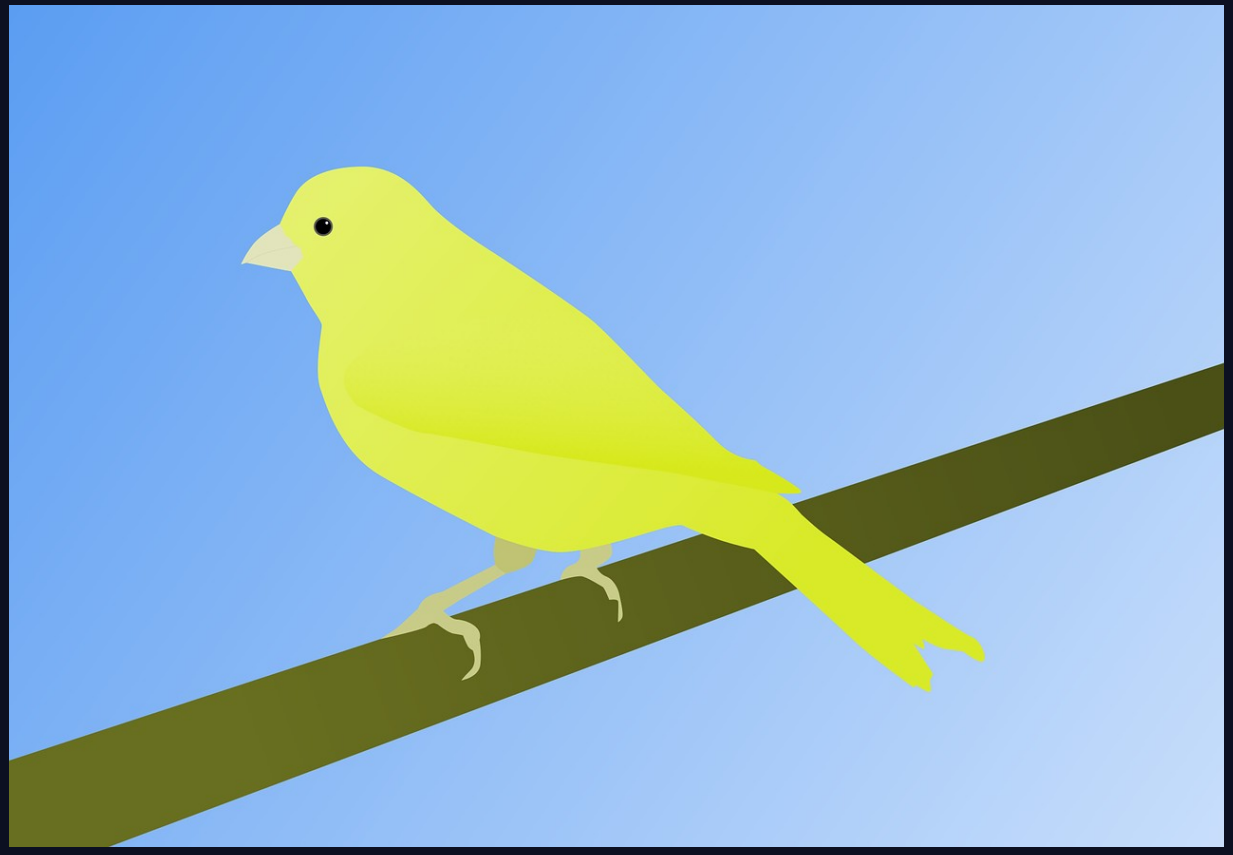
As one of the maintainers of the c-ares project I'm receiving mails for suspected security problems in c-ares and this was such a one. In this case, the email with said subject came from an individual who had reported a ChromeOS exploit to Google.

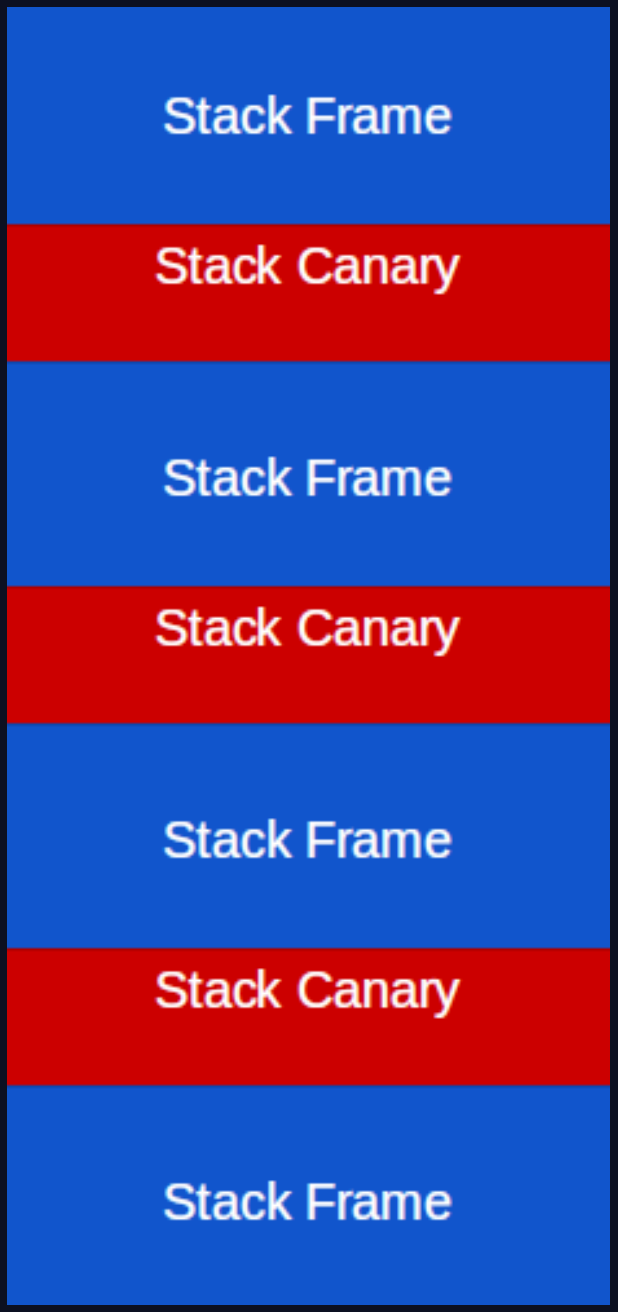
Triggered by a trailing escaped dot:

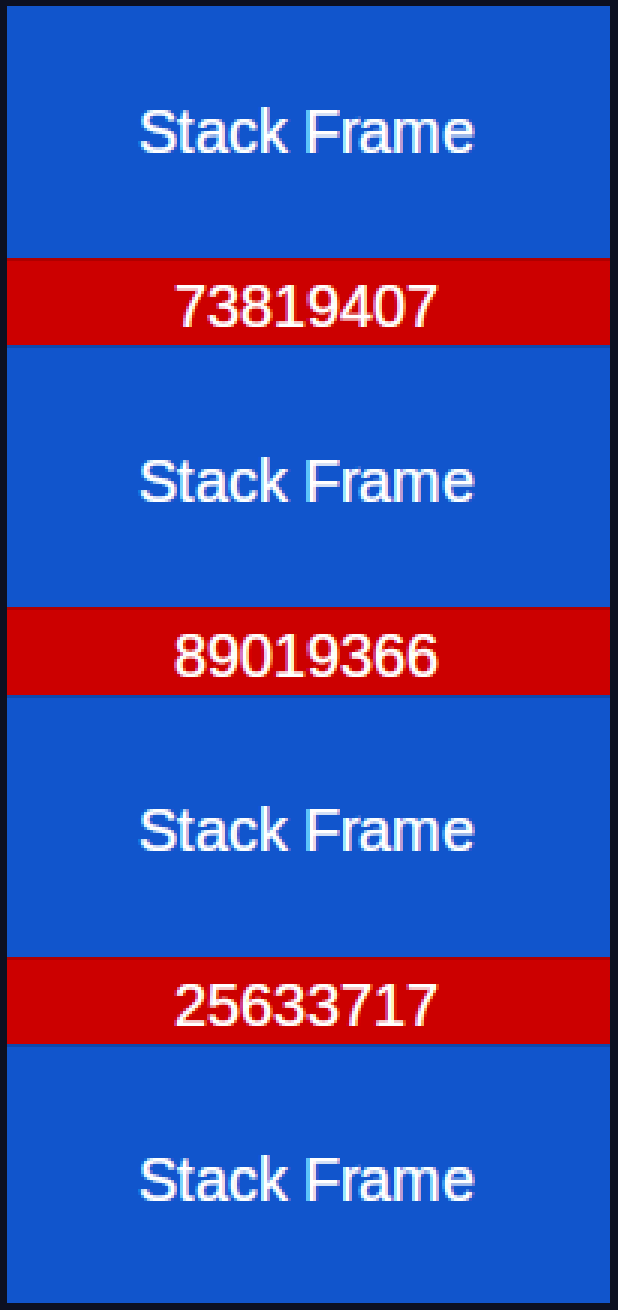
www.foo.com\.

Section 1c: Memory Corruption Mitigations

Stack Canaries



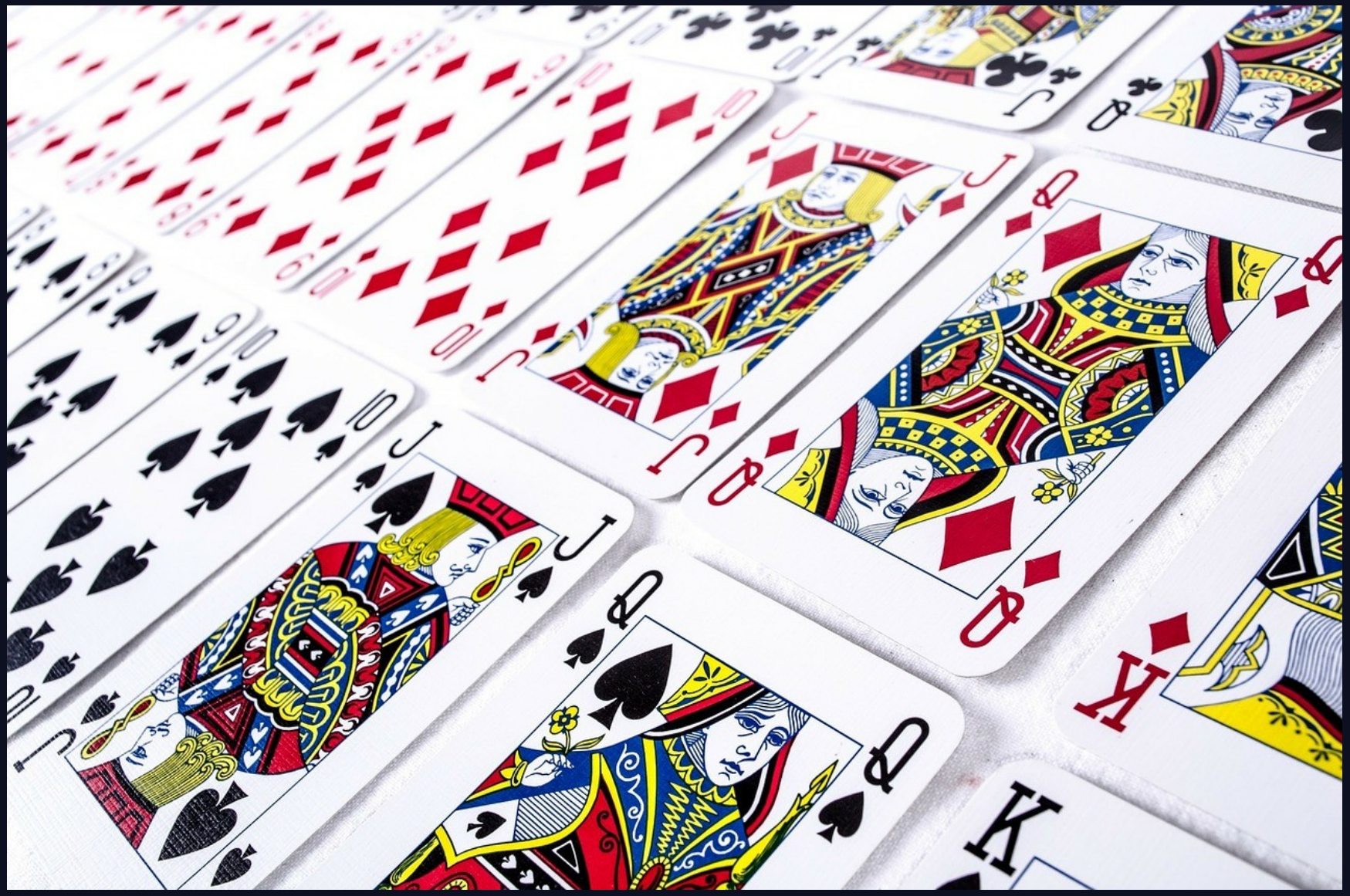

Stack Frame
Stack Canary
Stack Frame
Stack Canary
Stack Frame
Stack Canary
Stack Frame

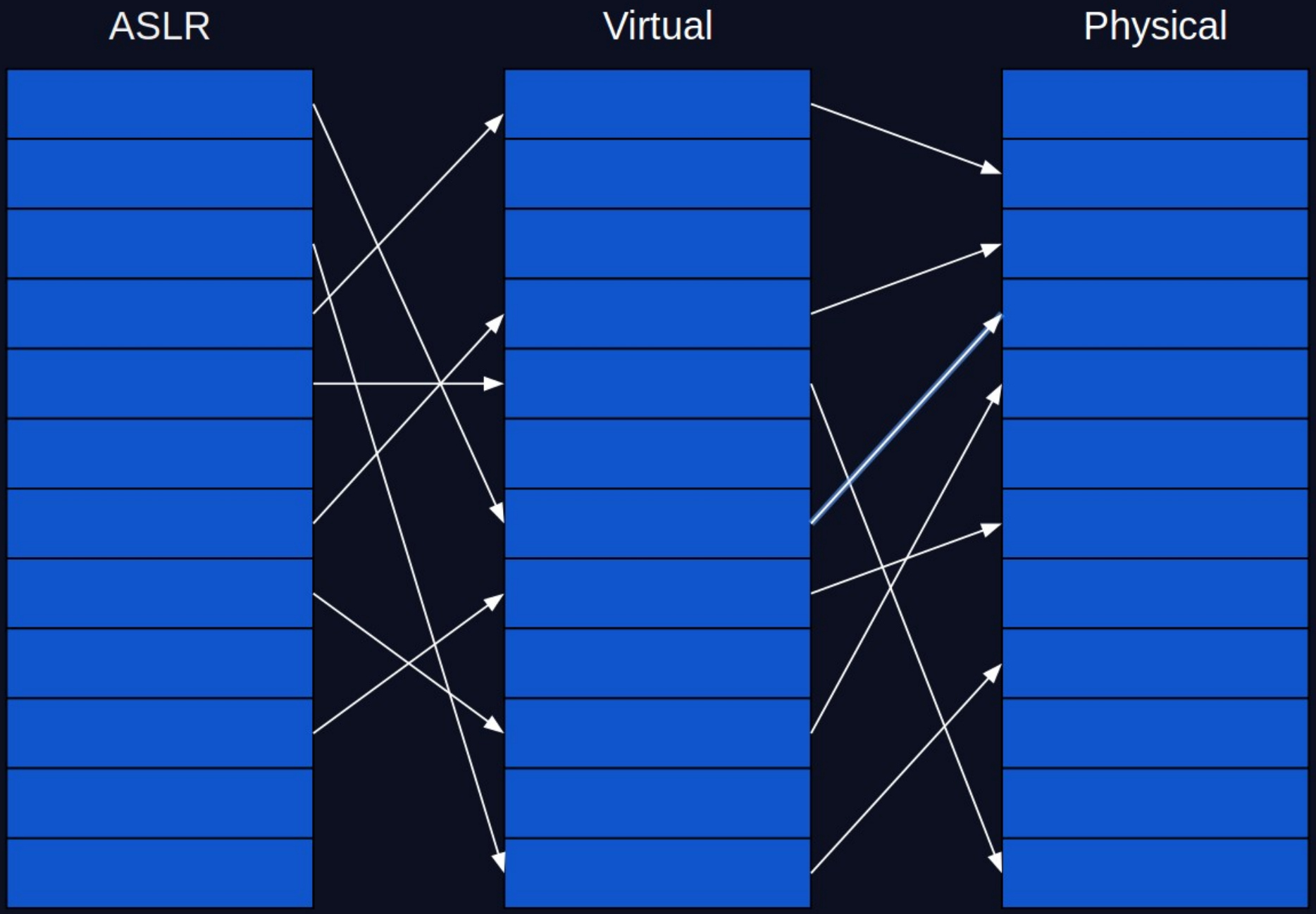

Stack Frame
73819407
Stack Frame
89019366
Stack Frame
25633717
Stack Frame

DEP
Data Execution Prevention

ASLR

Address Space Layout Randomization




ASLR
Virtual
Physical

Section 2:
A Post Fuzz Run Workflow

2a> Minimize crash corpus

2b> Use Memory Corruption Tools

2c> Determine Exploitability - or -
Find the Root Cause

Why minimize?

Minimize the Corpus of Crashes

Minimize each crashing case individually

fdupes


adrianlopezroche / fdupes
Watch 38
Code
Issues 32
Pull requests 11
Projects 0
Pulse
Graphs
FDUPES is a program for identifying or deleting duplicate files residing within specified directories.
C 87.7%
Groff 6.8%
Branch: master
New pull request
adrianlopezroche committed on GitHub Merge pull request #68 from falkartis/patch-1
Makefile.inc 1.6.1 release.
md5 Add modification note to md5/md5.c as required by license.
testdir fdupes-1.40

Section 2b: Memory Corruption Analysis Tools

All Bets are Off

Address Sanitizer

ASAN

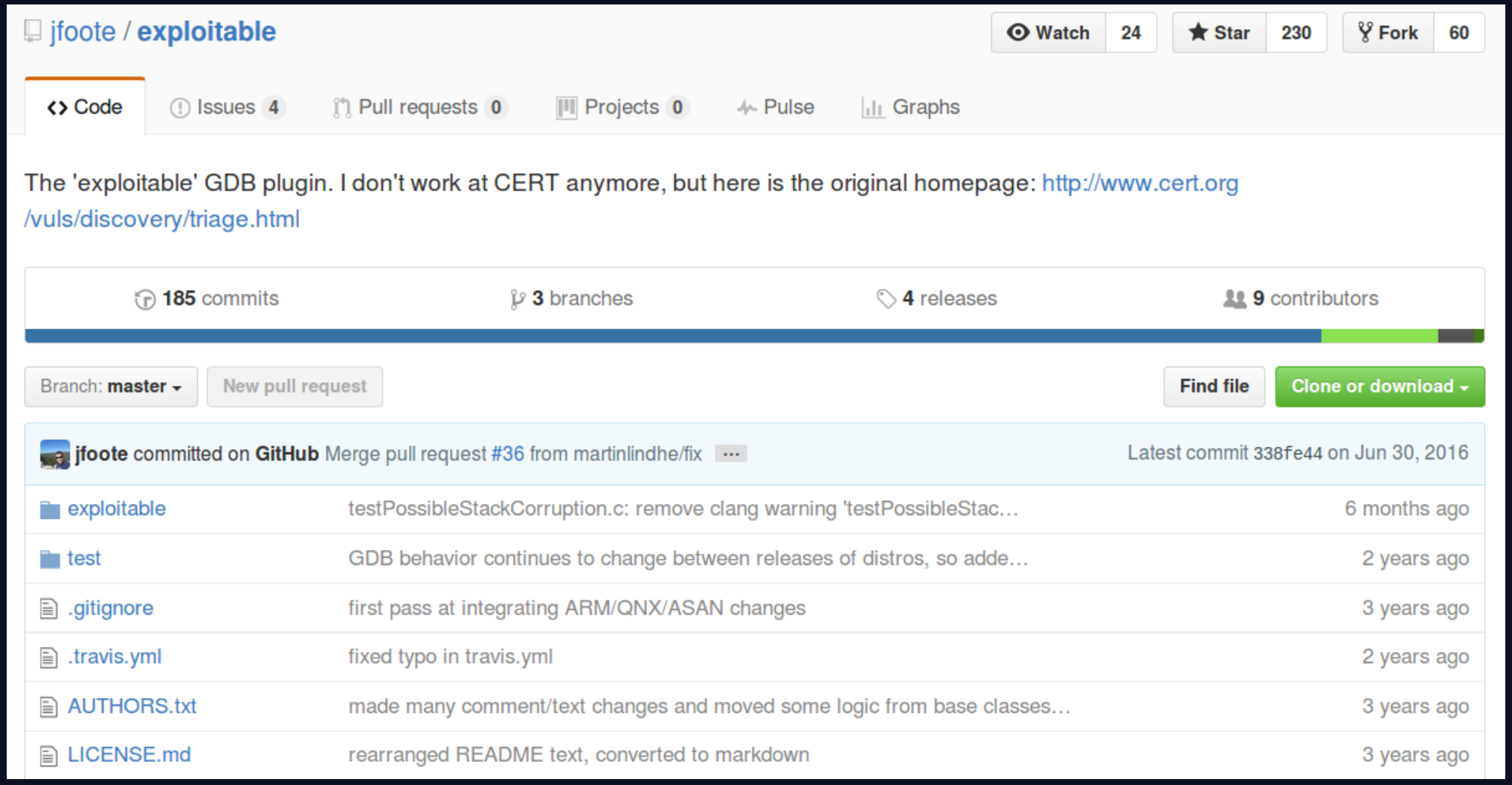
-fsanitize-address

```
==8670== ERROR: AddressSanitizer: heap-use-after-free on address 0x60080000bfd8 at pc
0x40bc17 bp 0x7ffd31b6df60 sp 0x7ffd31b6df58
READ of size 4 at 0x60080000bfd8 thread T0
    #0 0x40bc16 (/home/x/fuzzgoatASAN+0x40bc16)
    #1 0x40e169 (/home/x/fuzzgoatASAN+0x40e169)
    #2 0x401550 (/home/x/fuzzgoatASAN+0x401550)
    #3 0x7f5b3b2d1f44 (/lib/x86_64-linux-gnu/libc-2.19.so+0x21f44)
    #4 0x40173c (/home/x/fuzzgoatASAN+0x40173c)
0x60080000bfd8 is located 8 bytes inside of 40-byte region [0x60080000bfd0,0x60080000b
ff8)
freed by thread T0 here:
    #0 0x7f5b3b99033a (/usr/lib/x86_64-linux-gnu/libasan.so.0.0.0+0x1533a)
    #1 0x403778 (/home/x/fuzzgoatASAN+0x403778)
    #2 0x7f5b3e913d98 (/lib/x86_64-linux-gnu/ld-2.19.so+0x9d98)
previously allocated by thread T0 here:
    #0 0x7f5b3b9904e5 (/usr/lib/x86_64-linux-gnu/libasan.so.0.0.0+0x154e5)
    #1 0x402ef4 (/home/x/fuzzgoatASAN+0x402ef4)
    #2 0x7f5b3e913d98 (/lib/x86_64-linux-gnu/ld-2.19.so+0x9d98)
Shadow bytes around the buggy address:
  0x0c017fff97a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c017fff97b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c017fff97c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c017fff97d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c017fff97e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c017fff97f0: fa fa fa fa fa fa fa fa fa fa fd[fd]fd fd fd fa
```

Valgrind
(memcheck)

```
==8131== Memcheck, a memory error detector
==8131== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==8131== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==8131== Command: ./fuzzgoat deepCrash
==8131==
==8131== Invalid write of size 8
==8131==    at 0x403633: json_value_free_ex (fuzzgoat.c:965)
==8131==    by 0x40923C: json_value_free (fuzzgoat.c:1012)
==8131==    by 0x4010F1: main (main.c:166)
==8131==  Address 0x5502310 is 0 bytes inside a block of size 40 free'd
==8131==    at 0x4C2BDEC: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
)
==8131==    by 0x403588: new_value (fuzzgoat.c:123)
==8131==    by 0x4081E6: json_parse_ex (fuzzgoat.c:579)
==8131==    by 0x4091D9: json_parse (fuzzgoat.c:955)
==8131==    by 0x401098: main (main.c:156)
==8131==
==8131== Invalid read of size 4
==8131==    at 0x4037C4: json_value_free_ex (fuzzgoat.c:969)
==8131==    by 0x40923C: json_value_free (fuzzgoat.c:1012)
==8131==    by 0x4010F1: main (main.c:166)
==8131==  Address 0x5502318 is 8 bytes inside a block of size 40 free'd
```

Exploitable


jfoote / exploitable
Watch 24
Star 230
Fork 60
Code
Issues 4
Pull requests 0
Projects 0
Pulse
Graphs
The 'exploitable' GDB plugin. I don't work at CERT anymore, but here is the original homepage: http://www.cert.org/vuls/discovery/triage.html
185 commits
3 branches
4 releases
9 contributors
Branch: master
New pull request
Find file
Clone or download
jfoote committed on GitHub Merge pull request #36 from martinlindhe/fix
Latest commit 338fe44 on Jun 30, 2016
exploitable testPossibleStackCorruption.c: remove clang warning 'testPossibleStac... 6 months ago
test GDB behavior continues to change between releases of distros, so adde... 2 years ago
.gitignore first pass at integrating ARM/QNX/ASAN changes 3 years ago
.travis.yml fixed typo in travis.yml 2 years ago
AUTHORS.txt made many comment/text changes and moved some logic from base classes... 3 years ago
LICENSE.md rearranged README text, converted to markdown 3 years ago

```
[
    [
        "./fuzzgoat /home/fuzzstati0n/out/minimized_crashes/id:000000,sig:06,src:000050,op:havoc,rep:2,nde:node1",
        {
            "category": "EXPLOITABLE",
            "ranking": [
                10,
                22
            ],
            "explanation": "The target's backtrace indicates that libc has detected a heap error or that the target was executing a heap function when it stopped. This could be due to heap corruption, passing a bad pointer to a heap function such as free(), etc. Since heap errors might include buffer overflows, use-after-free situations, etc. they are generally considered exploitable.",
            "short_desc": "HeapError",
            "desc": "Heap error"
        }
    ]
]
```

Section 2c:
Determine Exploitability / Find the Root Cause

Disable ASLR

```
echo 0 | sudo tee
/proc/sys/kernel/randomize_va_space
```

Identify
critical memory
locations

gdb

gcc -g -O0 target.c

./target AAAA

0x41414141

rr

rr-project.org

rr github ci mailing list news #rr on irc.mozilla.org

what rr does

rr aspires to be your primary debugging tool, replacing — well, enhancing — gdb. You record a failure once, then debug the recording, deterministically, as many times as you want. The same execution is replayed every time.

rr also provides efficient reverse execution under gdb. Set breakpoints and data watchpoints and quickly reverse-execute to where they were hit.

It is OK and normal to:

Feel lost / frustrated
Take a lot of time
Feel like your wheels are spinning
Get sick of staring at hex

One More Thing:

Once the bugs are fixed - Fuzz the target again

Section 3:
Real World Examples

PHP:
Low invalid read

Bug #71683 Null pointer dereference in zend_hash_str_find_bucket

Submitted:	2016-02-28 15:02 UTC	**Modified:**	2016-03-11 23:20 UTC
From:	dmoorefo at gmail dot com	**Assigned:**	yohgaki
Status:	Closed	**Package:**	Reproducible crash
PHP Version:	7.0.3	**OS:**	Ubuntu 14.04.1 32-bit
Private report:	No	**CVE-ID:**	

Bug #71683 **Null pointer dereference in zend_hash_str_find_bucket**

Submitted:	2016-02-28 15:02 UTC	**Modified:**	2016-03-11 23:20 UTC
From:	dmoorefo at gmail dot com	**Assigned:**	yohgaki
Status:	Closed	**Package:**	Reproducible crash
PHP Version:	7.0.3	**OS:**	Ubuntu 14.04.1 32-bit
Private report:	No	**CVE-ID:**	

View | Add Comment | Developer | Edit

[2016-02-28 15:02 UTC] dmoorefo at gmail dot com

```
Description:
------------
A crafted ini file will cause a null pointer dereference leading to a segfault:

; crash.ini
session.auto_start=1
session.use_only_cookies=

It appears that setting session.auto_start makes php run as if in a web server context yet it is running
as a command line program. When REQUEST_URI is checked from the http globals, zend_hash_str_find_bucket
is called with a null hash table. This causes a 4 byte invalid read at 0x10.
```

```
Actual result:
--------------
==30268== Memcheck, a memory error detector
==30268== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al
==30268== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright
==30268== Command: php -c min-000000 -r
==30268==
==30268== Invalid read of size 4
==30268==    at 0x8450C37: zend_hash_str_find_bucket (zend_hash.c:515)
==30268==    by 0x8450C37: zend_hash_str_find (zend_hash.c:1959)
==30268==    by 0x829756B: php_session_start (session.c:1613)
==30268==    by 0x829B366: php_rinit_session (session.c:2624)
==30268==    by 0x829B385: zm_activate_session (session.c:2632)
==30268==    by 0x8438058: zend_activate_modules (zend_API.c:2536)
==30268==    by 0x83A06DF: php_request_startup (main.c:1626)
==30268==    by 0x8553892: do_cli (php_cli.c:945)
==30268==    by 0x8554A4C: main (php_cli.c:1345)
==30268==  Address 0x10 is not stack'd, malloc'd or (recently) free'd
```

[2016-03-09 10:14 UTC] yohgaki@php.net

```
This is patch fixes the crash.

diff --git a/ext/session/session.c b/ext/session/session.c
index 994d762..d0ee626 100644
--- a/ext/session/session.c
+++ b/ext/session/session.c
@@ -1611,6 +1611,7 @@ PHPAPI void php_session_start(void) /* {{{ */
                 * '<session-name>=<session-id>' to allow URLs of the form
                 * http://yoursite/<session-name>=<session-id>/script.php */
                if (PS(define_sid) && !PS(id) &&
+                       zend_is_auto_global_str("_SERVER", sizeof("_SERVER")-1) == SUCCESS &&
                        (data = zend_hash_str_find(Z_ARRVAL(PG(http_globals)[TRACK_VARS_SERVER]),
"REQUEST_URI", sizeof("REQUEST_URI") - 1)) &&
                        Z_TYPE_P(data) == IS_STRING &&
                        (p = strstr(Z_STRVAL_P(data), PS(session_name))) &&

As I commented earlier, it was jit global issue.
We may fix this as a reliability issue which is a part of security property, but I don't think this
crash is exploitable, is this?
```

Ruby: Heap Buffer Overflow

Ruby »

Ruby trunk

Search:

Overview | Activity | Roadmap | Issues | Wiki | Repository

Bug #12423

Regexp: Heap Buffer Overflow in regparse.c : next_state_value()

Added by David Moore 7 months ago. Updated 7 months ago.

Status:	Closed		
Priority:	Normal		
Assignee:	-		
Target version:	-		
ruby -v:	ruby 2.3.1p112 (2016-04-26 revision 54768) [i686-linux]	**Backport:**	2.1: WONTFIX, 2.2: DONE, 2.3: DONE

[ruby-core:75709]

Description

A crafted regular expression will cause a heap buffer overflow leading to invalid 4 byte reads/writes on 32-bit Ubuntu 14.04. The regular expression fails to close a character class and has an octal zero as the first character in the character class.

Despite the buffer overflow, ruby does not crash. This bug may have the same root cause as #12420.

```
==11873== ERROR: AddressSanitizer: heap-buffer-overflow on address 0xb470cd88
at pc 0xb730f9fd bp 0xbfc4d458 sp 0xbfc4d44c
READ of size 4 at 0xb470cd88 thread T0
    #0 0xb730f9fc (/usr/local/bin/ruby+0x1cb9fc)
    #1 0xb73116c9 (/usr/local/bin/ruby+0x1cd6c9)
    #2 0xb731a6fb (/usr/local/bin/ruby+0x1d66fb)
    #3 0xb731b856 (/usr/local/bin/ruby+0x1d7856)
    #4 0xb731baee (/usr/local/bin/ruby+0x1d7aee)
    #5 0xb731bdd9 (/usr/local/bin/ruby+0x1d7dd9)
    #6 0xb731c376 (/usr/local/bin/ruby+0x1d8376)
    #7 0xb72ccf09 (/usr/local/bin/ruby+0x188f09)
    #8 0xb729a77f (/usr/local/bin/ruby+0x15677f)
    #9 0xb729a8b6 (/usr/local/bin/ruby+0x1568b6)
    #10 0xb72a586e (/usr/local/bin/ruby+0x16186e)
    #11 0xb72a5bc1 (/usr/local/bin/ruby+0x161bc1)
    #12 0xb72a943c (/usr/local/bin/ruby+0x16543c)
```

```
==23692== Invalid write of size 4
==23692==    at 0x1C95F5: next_state_val (regparse.c:4478)
==23692==    by 0x1CA0D7: parse_char_class (regparse.c:4725)
==23692==    by 0x1CD36D: parse_exp (regparse.c:6187)
==23692==    by 0x1CD8F5: parse_branch (regparse.c:6365)
==23692==    by 0x1CD9BD: parse_subexp (regparse.c:6395)
==23692==    by 0x1CDB5D: parse_regexp (regparse.c:6443)
==23692==    by 0x1CDC80: onig_parse_make_tree (regparse.c:6485)
==23692==    by 0x1B27C6: onig_compile (regcomp.c:5739)
==23692==    by 0x1A0C20: onig_new_with_source (re.c:849)
==23692==    by 0x1A0CA8: make_regexp (re.c:873)
```

```
==23692== Invalid read of size 4
==23692==    at 0x1C95CB: next_state_val (regparse.c:4478)
==23692==    by 0x1CA0D7: parse_char_class (regparse.c:4725)
==23692==    by 0x1CD36D: parse_exp (regparse.c:6187)
==23692==    by 0x1CD8F5: parse_branch (regparse.c:6365)
==23692==    by 0x1CD9BD: parse_subexp (regparse.c:6395)
==23692==    by 0x1CDB5D: parse_regexp (regparse.c:6443)
==23692==    by 0x1CDC80: onig_parse_make_tree (regparse.c:6485)
==23692==    by 0x1B27C6: onig_compile (regcomp.c:5739)
==23692==    by 0x1A0C20: onig_new_with_source (re.c:849)
==23692==    by 0x1A0CA8: make_regexp (re.c:873)
```

Revision 55363
Added by Usaku NAKAMURA 7 months ago

merge revision(s) 55163,55165: [Backport #12420] [Backport #12423]

```
* regparse.c (fetch_token_in_cc): raise error if given octal escaped
  character is too big. [Bug #12420] [Bug #12423]

* re.c (unescape_nonascii): scan hex up to only 3 characters.
  [Bug #12420] [Bug #12423]
```

Netflix Dynomite: Invalid Write

Netflix Dynomite:

- Running in production ~2 years
- 1000 Customer facing nodes
- 1 Million ops/sec peak load

A dynomite admin can make a controlled 6 byte write to memory via a crafted dynomite.yml file resulting in heap corruption and possibly remote code execution and privilege escalation.

```
# xxd conf/dynomite.yml
0000000: 303a 0d20 303a 2022 4141 4141 4141 5c55   0:. 0: "AAAAAA\U
0000010: 3030 3042 3030 3030 3030 3030 3030 3030   000B000000000000
0000020: 305c 3022 0a                                0\0".
# gdb -q src/dynomite
Reading symbols from src/dynomite...done.
(gdb) r -t
Starting program: /root/dynomite/clean-dynomite/src/dynomite -t
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
*** Error in `/root/dynomite/clean-dynomite/src/dynomite': free(): invalid next size (fast):
0x080bd948 ***
Program received signal SIGABRT, Aborted.
0xb7fdccb0 in ?? ()
(gdb) x/x  0x080f5948
0x80f5948:    0x41414141
(gdb) x/x  0x080f594c
0x80f594c:    0xb0f24141
```

Netflix / **dynomite**

Fix security issue

```
We are using string functions which do the right thing during dup. So
lets use it. Otherwise we were writing an extra \0 sometimes causing
heap overflow
```

dev (#1) **v0.5.8** ··· alloc_msg_leak

```
@@ -75,14 +75,12 @@ string_duplicate(struct string *dst, const struct string *src)
75  75      ASSERT(dst->len == 0 && dst->data == NULL);
76  76      ASSERT(src->len != 0 && src->data != NULL);
77  77
78      -   dst->data = dn_strndup(src->data, src->len + 1);
    78  +   dst->data = dn_strndup(src->data, src->len);
79  79      if (dst->data == NULL) {
80  80          return DN_ENOMEM;
81  81      }
82  82
83      -   dst->len = src->len;
84      -   dst->data[dst->len] = '\0';
85      -
    83  +   dst->len = dn_strlen(dst->data);
```

References:

- RPI - Modern Binary Exploitation - GitHub: rpisec/mbe
- Hacking: The Art of Exploitation - Jon Erickson
- Project Zero Blog - What is Good Memory Corruption?
- Sean Heelan's Blog - Tracking Down Heap Overflows with rr

Thank You!

David Moore
@grajagandev
dave@fuzzstati0n.com